3.1 Problem-Solving Agents

Imagine an agent enjoying a touring vacation in Romania. The agent wants to take in the sights, improve its Romanian, enjoy the nightlife, avoid hangovers, and so on. The decision problem is a complex one. Now, suppose the agent is currently in the city of Arad and has a nonrefundable ticket to fly out of Bucharest the following day. The agent observes street signs and sees that there are three roads leading out of Arad: one toward Sibiu, one to Timisoara, and one to Zerind. None of these are the goal, so unless the agent is familiar with the geography of Romania, it will not know which road to follow.

1 We are assuming that most readers are in the same position and can easily imagine themselves to be as clueless as our agent. We apologize to Romanian readers who are unable to take advantage of this pedagogical device.

If the agent has no additional information—that is, if the environment is unknown—then the agent can do no better than to execute one of the actions at random. This sad situation is discussed in Chapter 4 . In this chapter, we will assume our agents always have access to information about the world, such as the map in Figure 3.1 . With that information, the agent can follow this four-phase problem-solving process:

GOAL FORMULATION: The agent adopts the goal of reaching Bucharest. Goals organize behavior by limiting the objectives and hence the actions to be considered.

Goal formulation

PROBLEM FORMULATION: The agent devises a description of the states and actions necessary to reach the goal—an abstract model of the relevant part of the world. For our agent, one good model is to consider the actions of traveling from one city to an adjacent city, and therefore the only fact about the state of the world that will change due to an action is the current city.

1 Problem formulation

SEARCH: Before taking any action in the real world, the agent simulates sequences of actions in its model, searching until it finds a sequence of actions that reaches the goal. Such a sequence is called a solution. The agent might have to simulate multiple sequences that do not reach the goal, but eventually it will find a solution (such as going from Arad to Sibiu to Fagaras to Bucharest), or it will find that no solution is possible.

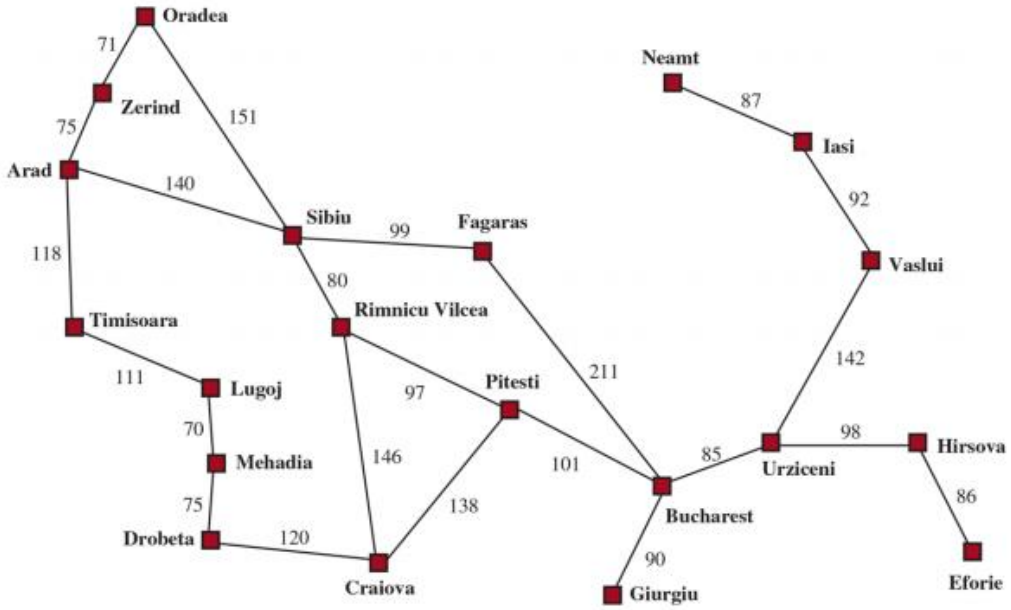
Search

Solution

EXECUTION: The agent can now execute the actions in the solution, one at a time.

Execution

Figure 3.1



A simplified road map of part of Romania, with road distances in miles.

A simplified road map of part of Romania, with road distances in miles.

It is an important property that in a fully observable, deterministic, known environment, the solution to any problem is a fixed sequence of actions: drive to Sibiu, then Fagaras, then Bucharest. If the model is correct, then once the agent has found a solution, it can ignore its percepts while it is executing the actions—closing its eyes, so to speak—because the solution is guaranteed to lead to the goal. Control theorists call this an open-loop system: ignoring the percepts breaks the loop between agent and environment. If there is a chance that the model is incorrect, or the environment is nondeterministic, then the agent would be safer using a closed-loop approach that monitors the percepts (see Section 4.4).

Open-loop

Closed-loop

In partially observable or nondeterministic environments, a solution would be a branching strategy that recommends different future actions depending on what percepts arrive. For example, the agent might plan to drive from Arad to Sibiu but might need a contingency plan in case it arrives in Zerind by accident or finds a sign saying “Drum Închis” (Road Closed).

3.1.1 Search problems and solutions

A search problem can be defined formally as follows:

Problem

A set of possible states that the environment can be in. We call this the state space.

States

State space

The initial state that the agent starts in. For example: Arad.

Initial state

A set of one or more goal states. Sometimes there is one goal state (e.g., Bucharest), sometimes there is a small set of alternative goal states, and sometimes the goal is defined by a property that applies to many states (potentially an infinite number). Forexample, in a vacuum-cleaner world, the goal might be to have no dirt in any location, regardless of any other facts about the state. We can account for all three of these possibilities by specifying an IS-GOAL method for a problem. In this chapter we will sometimes say “the goal” for simplicity, but what we say also applies to “any one of the possible goal states.”

Goal states

The actions available to the agent. Given a state ACTIONS returns a finite set of actions that can be executed in We say that each of these actions is applicable in An example:

2 For problems with an infinite number of actions we would need techniques that go beyond this chapter.

ACTIONS(Arad) = {ToSibiu, ToTimisoara, ToZerind}.

Action

Applicable

A transition model, which describes what each action does. RESULT returns the state that results from doing action in state For example,

RESULT(Arad, ToZerind) = Zerind.

Transition model s

An action cost function, denoted by ACTION-COST when we are programming or when we are doing math, that gives the numeric cost of applying action in state to reach state A problem-solving agent should use a cost function that reflects its own performance measure; for example, for route-finding agents, the cost of an action might be the length in miles (as seen in Figure 3.1), or it might be the time it takes to complete the action.

Action cost function

A sequence of actions forms a path, and a solution is a path from the initial state to a goal state. We assume that action costs are additive; that is, the total cost of a path is the sum of the individual action costs. An optimal solution has the lowest path cost among all solutions. In this chapter, we assume that all action costs will be positive, to avoid certain complications.

3 In any problem with a cycle of net negative cost, the cost-optimal solution is to go around that cycle an infinite number of times. The Bellman–Ford and Floyd–Warshall algorithms (not covered here) handle negative-cost actions, as long as there are no negative cycles. It is easy to accommodate zero-cost actions, as long as the number of consecutive zero-cost actions is bounded. For example, we might have a robot where there is a cost to move, but zero cost to rotate 90°; the algorithms in this chapter can handle this as long as no more than three consecutive 90° turns are allowed. There is also a complication with problems that have an infinite number of arbitrarily small action costs. Consider a version of Zeno's paradox where there is an action to move half way to the goal, at a cost of half of the previous move. This problem has no solution with a finite number of actions, but to prevent a search from taking an unbounded number of actions without quite reaching the goal, we can require that all action costs be at least for some small positive value

Path

Optimal solution

The state space can be represented as a graph in which the vertices are states and the directed edges between them are actions. The map of Romania shown in Figure 3.1 is such a graph, where each road indicates two actions, one in each direction.

Graph

3.1.2 Formulating problems Our formulation of the problem of getting to Bucharest is a model—an abstract mathematical description—and not the real thing. Compare the simple atomic state description Arad to an actual cross-country trip, where the state of the world includes so many things: the traveling companions, the current radio program, the scenery out of the window, the proximity of law enforcement officers, the distance to the next rest stop, the condition of the road, the weather, the traffic, and so on. All these considerations are left out of our model because they are irrelevant to the problem of finding a route to Bucharest. The process of removing detail from a representation is called abstraction. A good problem formulation has the right level of detail. If the actions were at the level of “move the right foot forward a centimeter” or “turn the steering wheel one degree left,” the agent would probably never find its way out of the parking lot, let alone to Bucharest.

Abstraction

Can we be more precise about the appropriate level of abstraction? Think of the abstract states and actions we have chosen as corresponding to large sets of detailed world states and detailed action sequences. Now consider a solution to the abstract problem: for example, the path from Arad to Sibiu to Rimnicu Vilcea to Pitesti to Bucharest. This abstract solution corresponds to a large number of more detailed paths. For example, we could drive with the radio on between Sibiu and Rimnicu Vilcea, and then switch it off for the rest of the trip.

Level of abstraction

The abstraction is valid if we can elaborate any abstract solution into a solution in the more detailed world; a sufficient condition is that for every detailed state that is “in Arad,” there is a detailed path to some state that is “in Sibiu,” and so on. The abstraction is useful if carrying out each of the actions in the solution is easier than the original problem; in our case, the action “drive from Arad to Sibiu” can be carried out without further search or planning by a driver with average skill. The choice of a good abstraction thus involves removing as much detail as possible while retaining validity and ensuring that the abstract actions are easy to carry out. Were it not for the ability to construct useful abstractions, intelligent agents would be completely swamped by the real world.